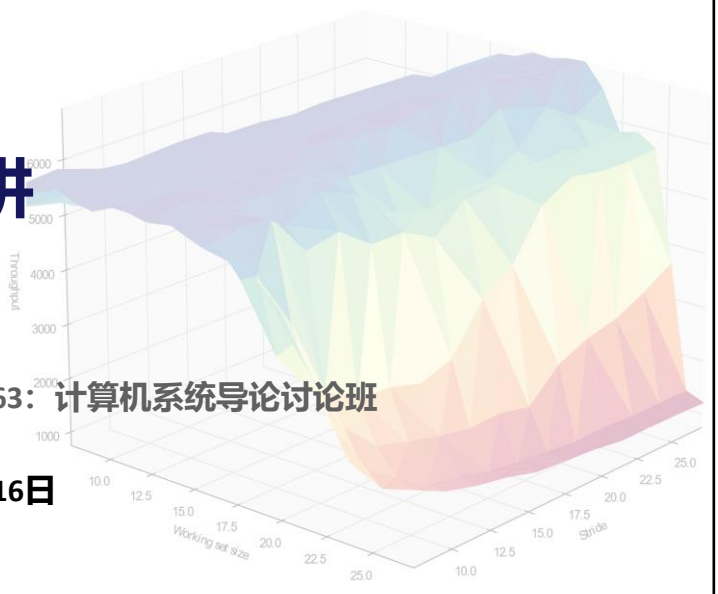
第七讲



PKU 04832363：计算机系统导论讨论班

王畅

2021年11月16日

重点细节

- **P501页上部，异常控制流的定义！**
  - 控制流是PC序列；跳转和返回是控制流突变；异常控制流是因为异常而发生的跳转和返回。
- **异常是软硬件结合实现的（8.1开头）**
  - 大致地说，发生异常后跳到处理程序这件事一般是硬件完成（异常表、异常号、异常基址寄存器），而异常处理程序本身是操作系统的一部分。整个过程要会复述。
- **异常和过程调用的区别（理解并记忆）**
  - 特别是返回的位置，还有内核模式/内核栈
  - ~~操作系统也要考~~
- **异常的类别**
  - 表8.4要理解，而且每种异常要掌握一个例子，例如图8.9。
- **系统调用的特点和惯例（P506下面，特别是寄存器使用）**

2

异常表是系统启动时操作系统负责填写的。

异常vs调用

调用	异常
返回地址一定是下一条指令	返回地址可能是当前指令，可能是下一条，也可能不返回
只把参数和返回地址压入用户栈中	会把恢复中断状态所需的额外的处理器状态（如EFLAGS）压入内核栈中
运行在用户模式	运行在内核模式（超级用户模式）

考点1a：异常的基本概念

- **可以用操作系统的往年考题复习：复述系统调用的整个过程，自己回忆有哪些注意点！**

- **举例：（但ICS绝对不是这样考的）**
  - 一个支持100个system call的操作系统是如何实现的？
  - 简述中断异常区别。
  - 简述内核处理被触发和内核处理流程。
  - 简述IA32系统调用流程。
  - 简述设计增加系统调用过程。
  - 可以参考https://ivenwang.com/2019/11/11/os-3/复习（不用记忆，就是看一看）

概念辨析

- **学完本课程后，几位同学聚在一起讨论有关异常的话题，请问你认为他们中谁学习的结果有错误?**
  - A．发生异常和异常处理意味着控制流的突变。
  - B．与异常相关的处理是由硬件和操作系统共同完成的。
  - C．异常是由于计算机系统发生了不可恢复的错误导致的。
  - D．异常的发生可能是异步的，也可能是同步的。

5

C，中止是不可恢复的错误，故障是可能恢复也可能不恢复的错误。

进程的基本概念

- **进程的哪些东西是独立的？P508**
  - 上下文：代码、数据、**栈、寄存器、PC**、环境变量和打开文件
  - 进程私有地址空间的结构：注意内核栈（P510）
- **并发和并行的区别和联系P509**
  - 假并行是并发但不是并行
  - 并行一定是并发
- **进程调度、抢占的概念**
  - 上下文切换（切换上面的上下文，转到不同的地方运行，是实现多任务的手段）
  - 用户模式和内核模式（由硬件实现，软件辅助）
  - 内核不是一个进程

进程管理函数

- **这里函数非常多，建议亲自写程序体会，也可以用man命令查手册**
  - getpid()返回值是有符号整数
  - fork()函数是重难点。第一个是它返回两次，父进程返回PID，子进程返回0。第二个是子进程是父进程的一份独立的副本，但打开文件表指向的地方是一样的（后面会学到）。
  - waitpid()是另一个重难点。僵尸子进程原则上要回收；如果子进程尚未结束而父进程结束了，则它们都变成孤儿，由init代管。
  - execve执行成功时不返回（失败还是返回）。

Waitpid()的使用模式值得注意。通常传入的options由一个整数表示，这个整数由多个常量掩码用或运算合并。

Activity 1

轮流回答问题

概念辨析

- **区分下列异常的类型：**

- **执行指令`mov $57, %eax; syscall`**
- **程序执行过程中，发现它所使用的物理内存损坏了**
- **程序执行过程中，试图往`main`函数的内存中写入数据**
- **按下键盘**
- **磁盘读出了一块数据**
- **用户程序执行了指令`lgdt`，但是这个指令只能在内核模式下执行**

陷阱（系统调用，人为的异常）

中止

故障（触发系统的保护机制一般都是故障）

中断

中断（注意是磁盘操作结束，要发中断信号）

故障

考点1b：常见系统API的使用

- **熟记8.4整节的各种API用法**
  - 参数还有返回值
  - 各种常量的使用和含义
- **必要时实验加深印象**

概念辨析

- **关于进程，以下说法正确的是：**
  - A. 没有设置模式位时，进程运行在用户模式中，允许执行特权指令，例如发起I/O操作。
  - B. 调用waitpid(-1, NULL, WNOHANG & WUNTRACED)会立即返回：如果调用进程的所有子进程都没有被停止或终止，则返回0；如果有停止或终止的子进程，则返回其中一个的ID。
  - C. execve函数的第三个参数envp指向一个以null结尾的指针数组，其中每一个指针指向一个形如name=value的环境变量字符串。
  - D. 进程可以通过使用signal函数修改和信号相关联的默认行为，唯一的例外是SIGKILL，它的默认行为是不能修改的。

C。A显然是错误的，特权指令必须在内核模式下进行。B中option的位应该用|运算符来连结，而不是且，这样一般会得到0。D不是唯一的例外，SIGKILL和SIGSTOP都是例外。

Activity 2

问题求解

考点0：结合汇编的符号解析/重定位分析

- 做法参看上一讲
- Section 12 练习2、5、7、10、12

13

本节需要补充有关动态链接的习题，请参看下一次。

考点2：Fork

- **理解子进程是父进程的一个独立副本**
  - `fork`瞬间二者是一模一样的，后面则可能不同
  - 子进程从`fork`返回处开始执行，因此`fork`会返回两次。
- **会画进程图，并能快速进行拓扑排序**
  - 不要用书上的那种图，建议使用更加简明的画法
  - 在树上拓扑排序类似于写出所有“前序”遍历序列

Fork Puzzles

- **下面的程序中，父进程的输出是什么，子进程的输出是什么?**

```
int main()
{
    int a = 9;
    if (Fork() == 0)
        printf("p1: a=%d\n", a--);
    printf("p2: a=%d\n", a++);
    exit(0);
}
```

15

Fork在子进程中返回0，所以父进程输出9，子进程输出9、8。这里要注意的是自增自减的性质。

Fork Puzzles

- **下列程序可能的输出是：**

A. 1 2 -1 0
B. 0 0 -1 1
C. 1 -1 0 0
D. 0 -1 1 2

```
int count = 0;
int pid = fork();
if (pid == 0) {
    printf("count = %d\n", --count);
} else {
    printf("count = %d\n", ++count);
}
printf("count = %d\n", ++count);
```

16

A。首先父子进程的数据是相互独立的，所以父进程输出1、2而子进程输出-1、0，故B、C不对。另外每个进程自己输出的顺序是确定的，所以D不对。

Fork Puzzles

```
int main() {
    char c = 'A';
    printf("%c", c);
    fflush(stdout);
    if (fork() == 0) {
        c++;
        printf("%c", c);
        fflush(stdout);
    }
    else {
        printf("%c", c);
        fflush(stdout);
        fork();
    }
    c++;
    printf("%c", c);
    fflush(stdout);
    return 0;
}
```

- **阅读左边的程序，下列哪些输出是可能的?**

- **AABBBC** Y
- **ABCABB** Y
- **ABBABC** N
- **AACBBC** N
- **ABABCB** Y
- **ABCBAB** N

采用画进程之间有向关系图并拓扑排序的方法即可（注意不要用书上的画法）。

Fork Puzzles

```c
int main() {
    int child_status;
    char c = 'A';
    printf("%c", c);
    fflush(stdout);
    c++;
    if (fork() == 0) {
        printf("%c", c);
        fflush(stdout);
        c++;
        fork();
    }
    else {
        printf("%c", c);
        fflush(stdout);
        c += 2;
        wait(&child_status);
    }
    printf("%c", c);
    fflush(stdout);
    exit(0);
}
```

- **阅读左边的程序，下列哪些输出是可能的?**

- **ABBCCD** **Y**
- **ABBCDC** **Y**
- **ABBDCC** **N**
- **ABDBCC** **N**
- **ABCDBC** **N**
- **ABCDCB** **N**

18

当需要wait时，要在关系图中，子进程结束前的最后一个输出向父进程wait后的第一个输出连一条有向边，然后再拓扑排序。

Craaaaaaaazy Fork Puzzles（了解）

- 以下程序的运行结果输出几个1?

```
int main() {
    if (fork() || fork())
        fork();
    printf("1 ");
    return 0;
}
```

19

5个1。

1. It will create two process one parent P (has process ID of child process)and other is child C1 (process ID = 0).

2. In if statement we used OR operator(||) and in this case second condition is evaluated when first condition is false.

3. Parent process P will return positive integer so it directly execute statement and create two more processes (one parent P and other is child C2). Child process C1 will return 0 so it checks for second condition and second condition again create two more processes(one parent C1 and other is child C3).

4. C1 return positive integer so it will further create two more processes (one parent C1 and other is child C4). Child C3 return 0 so it will directly

```
print 1.
```

Craaaaaaaazy Fork Puzzles（了解）

- 以下程序的运行结果输出几个2?

```
int main() {
    if (fork() && (!fork())) {
        if (fork() || fork()) {
            fork();
        }
    }
    printf("2 ");
    return 0;
}
```


7个2。

1. Fork will create two process one parent P (has process id of new child) and other one is child C1 (process id=0).

2. In if statement we are using AND operator (i.e, &&) and in this case if first condition is false then it will not evaluate second condition and print 2. Parent process P check for second condition and create two new processes (one parent P and other is child C2). In second condition we are using NOT operator which return true for child process C2 and it executes inner if statement.

3. Child C2 again create two new processes (one parent C2 and child C3) and we are using OR operator (i.e, ||) which evaluate second condition when first condition is false. Parent C2 execute if part and create two new processes (one parent C2

and child C4) whereas child C3 check for second condition and create two new processes (one parent C3 and child C5).

4. Parent C3 enters in if part and further create two new processes (one parent C3 and child C6).

Craaaaaaaazy Fork Puzzles（了解）

- 以下程序的运行结果可能有哪些?

```
int main() {
    if (fork()) {
        if (!fork()) {
            fork();
            printf("1 ");
        }
        else {
            printf("2 ");
        }
    }
    else {
        printf("3 ");
    }
    printf("4 ");
    return 0;
}
```

21

1. It will create two process one parent P (has process ID of child process) and other is child C1 (process ID = 0).

2. When condition is true parent P executes if statement and child C1 executes else statement and print 3. Parent P checks next if statement and create two process (one parent P and child C2). In if statement we are using not operator (i.e, !), it executes for child process C2 and parent P executes else part and print value 2. Child C2

further creates two new processes (one parent C2 and other is child C3).

There’re many possible outputs, one of them being 2 4 1 4 1 4 3 4.

any questions?

Thanks & 感谢观看